## Automated Servicing of Agents

Brazier, F.M.; Wijngaards, N.J.E.

***published in***
AISB Journal
2001

***document version***
Publisher's PDF, also known as Version of record

*AISB Journal* **1**(1), © 2001
http://www.aisb.org.uk

# Automated Servicing of Agents

Frances M. T. Brazier and Niek J. E. Wijngaards

Intelligent Interactive Distributed Systems Group,
Faculty of Sciences, Vrije Universiteit Amsterdam,
de Boelelaan 1081a; 1081 HV Amsterdam, The Netherlands
*frances@cs.vu.nl ; niek@cs.vu.nl*

**Abstract**

Agents need to be able to adapt to changes in their environment. One way to achieve this, is to service agents when needed. A separate servicing facility, an agent factory, is capable of automatically modifying agents. This paper discusses the feasibility of automated servicing.

## 1 Introduction

Agents typically operate in dynamic environments. Agents come and go, objects appear and disappear, and cultures and conventions change. Whenever an environment of an agent changes to the extent that an agent is unable to cope with (parts of) the environment, an agent needs to adapt. Changes in the social environment of an agent, for example, may require modifications to existing agents. A new agent communication language, or new protocols for auctions, are examples of such changes. An agent may be able to detect gaps in its abilities; it may not be able to fill these gaps with its own built-in learning mechanisms. Whether the need for servicing is detected by an agent itself, or by another agent (automated or human) is irrelevant to the concept involved: external assistance may be needed to perform the necessary modifications.

This paper discusses the feasibility of a service for automated revision. In Section 2, needs for adaptation are discussed. An automated servicing facility, an agent factory, is described in Section 3. An example of adapting an agent, based on an existing prototype automated servicing service, is provided in Section 4. The feasibility of such a service for automated revision is discussed in Section 5, in which the agent factory is also compared to related approaches. The results presented in this paper are discussed in Section 6.

## 2 Adaptive Agents

Both static and mobile agents may encounter the need for adaptation. In this section an example is used to illustrate a few situations in which external adaptation is feasible.

The focus in this example is on an information gathering agent. The information gathering agent is assumed to be mobile. Its task is to find information for a researcher about travel arrangements needed to attend a conference. To this purpose the agent communicates with three other agents (a personal assistant agent, a travel agent, and a bank agent) and interacts with the World-Wide Web.

**Example 1.** The personal assistant agent informs the information gathering agent about its preferences with respect to travel agents, and about the researcher's travel preferences. The personal assistant agent has acquired some of this information directly from the researcher, and has acquired some over the course of time from the researcher and from its own experience. The information gathering agent maintains a profile of the personal assistant agent, and adapts this profile on the basis of interaction with the personal assistant agent (e.g., as also encountered in negotiation settings (Bui et al., 1996)). Note that in this example personification is not aimed at personalising an agent's representation of a human user (e.g., see (Soltysiak and Crabtree, 1998; Wells and Wolfers, 2000)), but the profile of the personal assistant agent.

**Example 2.** The information gathering agent consults the World-Wide Web to find dates and a location for the aforementioned conference. The conference page is annotated in an ontology that is unfamiliar to the agent. For example, OIL (Fensel et al., 2000; Horrocks et al., 2001) has been used instead of XML (Bray et al., 2000). One way to approach this problem is to have the information gathering agent acquire understanding of this ontology. Another option is to use an intermediary agent (e.g., brokers/matchmakers (Wong and Sycara, 2000)) to find an agent capable of translating between ontologies, e.g. via SOAP, (the Simple Object Access Protocol (Box et al., 2000)). In this last case the agent needs to "travel" and collaborate with its new assistant during interaction with the conference site.

The information gathering agent also wishes to discuss possible travel arrangements with the travel agent. The travel agent indicates that it only "speaks" a specific language and protocol. The information gathering agent needs to acquire this agent language and protocol. This is comparable to the acquisition of a new ontology sketched above.

**Example 3.** During the discussion between the information gathering agent and the travel agent, the issue of credit rating arises. The information gathering agent needs to prove to the travel agent that it is trustworthy and has an acceptable credit rating. In addition to security clearance on its own trustworthiness, the information gathering agent needs additional information from the personal assistant agent. The information gathering agent needs permission to ask a bank for this information and the name and address of a specific bank agent. The bank agent, in turn, requires certificates and a guarantee from the information gathering agent that specific security measures are in place, before it will provide any other information. If the information gathering agent does not have this functionality it may be possible to add this functionality to the agent. (Please note that adding functionality may not be the only measure that needs to be taken in this case.)

In each of the situations sketched above an automated servicing process is to be used. The types of adaptation involved are:

- Personalisation: an agent can be provided with profiles specific to its current co-operation partners.
- Domain and languages: an agent can be adapted to include knowledge about a specific domain to understand a specific agent communication language and protocol.
- Functionality: new functionality or characteristics can be added to (or deleted from) an agent.

# 3 An agent factory

An automated agent servicing facility, an agent factory, is described in this section. The agent factory, in essence, re-designs descriptions of agents. Previous research (Brazier et al., 2000a; Brazier et al., 2000b) focussed on automated redesign of multi-agent systems at a detailed (conceptual) level. The automated servicing service described in this paper is an extension of this work in two ways.

The first distinction with the previous work is that the agent factory as presented in this paper is not primarily focussed on re-designing agents on the basis of first principles on a conceptual level, as described in (Brazier et al., 2000b). The agent factory uses building blocks to construct, and adapt, agents. Building blocks can be templates, i.e. skeletons that describe the architecture of a (larger) part of an agent. Components are building blocks with specific functionality. Templates and components are combined according to pre-defined rules.

The second distinction with previous work is a broadening of the scope of the re-design process. The agent factory modifies not only the conceptual description of an agent, but also its operational code. This necessitates knowledge about the relationship between the conceptual description and detailed (operational) description of templates and components.

On the basis of a need for adaptation, the automated servicing process re-configures templates and components at both levels. Re-configuration (an instance of a re-design process) of an agent first takes place at the conceptual level: templates and components are removed and added until a satisfactory conceptual agent description is acquired. On the basis of the configuration of templates and components in the conceptual description of an agent a detailed (operational) description of an agent is generated.

To facilitate the automated re-design of agents, a number of assumptions have been made on the descriptions of an agent (Section 3.1). In addition, the agent factory has a library of building blocks, the so-called templates and components (Section 3.2). The configuration task of the agent factory (Section 3.5) is based on knowledge of the characteristics and properties (Section 3.3), and the availability of templates and components (Section 3.4).

## 3.1 Assumptions on the design of agents

The feasibility of an automated service for revision of agents depends largely on the assumptions imposed on the design of the agents. The most important underlying assumptions for an agent adaptation service used in this paper are as follows.

The first assumption is that agents have a compositional structure. A compositional structure greatly facilitates the possibilities of adding, removing and changing parts of an agent. This principle is used throughout software design, ranging from describing processes (e.g., JSD (Jackson, 1975)), via object-oriented programming (e.g., (Booch, 1991; Pressman, 1997; Wieringa, 1996)) to component-based programming (e.g., (Hopkins, 2000)).

The second assumption is that re-usable parts of agents can be identified: templates (i.e., skeletons) and components (i.e., building blocks). The agent factory can build an agent by correctly configuring templates and components. This assumption relates to design patterns (e.g., (Gamma et al., 1994; Peña-Mora and Vadhavkar, 1996; Riel, 1996)) and libraries of software with specific functionality (e.g., problem-solving models (Schreiber et al., 1999) or generic task models (Brazier et al., 1998)).

The third assumption is that templates and components are described at two levels of abstraction: a conceptual description and a detailed description. This assumption circumvents two problems. On the one hand, it is difficult to determine a conceptual description on the basis of a detailed, operational description of (part of) a system (e.g., (Jackson, 1995)). On the other hand, it is again difficult to determine the operational description of (part of) a system on the basis of a conceptual description (e.g., (Rumbaugh et al., 1999)). In the case of the agent factory, the detailed description is also an operational description.

The fourth assumption is that properties and knowledge of properties are available to describe templates and components. Interfaces provided and required by templates and components need to be described (e.g., as is done in work on describing classes of diagnostic (non-user-interactive) problem-solving methods by (Benjamins, 1995)).

A fifth assumption is that no commitments have been made to specific languages and/or ontologies. The languages used for the descriptions of templates and components on both levels of abstraction are left open, as are the descriptions, and contents, of the properties and knowledge on properties to describe templates and components. The agent factory is explicitly developed to be an open architecture.

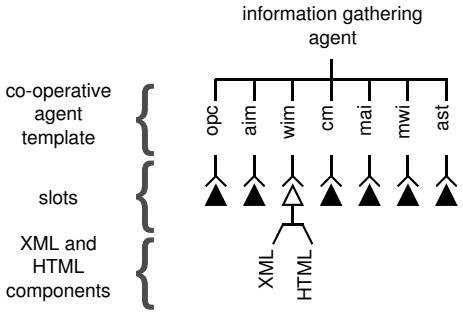


Figure 1: Graphical representation of templates and components and their slots.

## 3.2 Templates and components

Templates and components are the building blocks with which agents are constructed. Templates are skeletons which describe an architecture of a (larger) part of an agent. A template is usually combined with a number of (other) templates and/or components. A component is a building block with specific functionality.

For each conceptual description, a number of detailed, operational descriptions may be devised. These operational descriptions may differ in the operational language (e.g., C, C++, Java), but also in, for example, the efficiency of the operational code.

Templates and components are configurable. However, templates or components cannot be combined indiscriminately. The open slot concept is used to regulate the ways in which templates and components may be combined. An open slot in a template or component has associated properties that prescribe the properties of the entity to be 'inserted' in addition to the interface of the required building block.

A mapping relation is defined between building blocks containing conceptual descriptions and building blocks containing detailed descriptions. Each conceptual building block may be related to a number of detailed building blocks; the inverse may hold as well.

Templates specify the architecture of an information gathering agent. In figure 1, the information gathering agent is shown to consist of seven processes (as explained in Section 4). Each of these processes has a slot, which is filled by a combination of templates and/or components. The open slot for the world interaction management process (wim), is shown to be filled with two components, which provide specific functionality to interact with web pages annotated in HTML and XML.

An "open-slot preserving" relationship is defined in the mapping relation between building blocks, so that each open slot in a conceptual template or component is related to an open slot in the associated detailed template or component. The open-slot preserving relationship between related conceptual and operational building blocks implies that templates and components are combined in the same configuration at both levels of abstraction. The two-stage revision process facilitates the generation of operational code: on the basis of the configuration of templates and components at a conceptual level, the detailed, operational code is generated in a relatively straightforward manner, as explained in the next section.

## 3.3 Details of templates and components

The building blocks used by the agent factory, templates and components, have the same structure, as depicted in figure 2. This structure does not make a commitment to specific conceptual or detailed (operational) description languages, but includes types of information that are also included in structures designed to describe design patterns (e.g., (Gamma et al., 1994)).

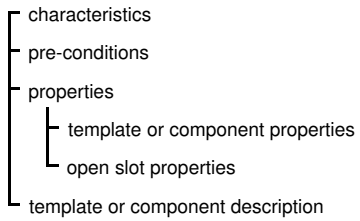


Figure 2: Structure of templates and components used by the agent factory.

The characteristics of a building block describe its name, creation dates, authors, version information, and level of abstraction. This information is not related to the description inside the building block.

The pre-conditions contain assumptions and requirements of the interface of the building block that have to be satisfied by the environment (i.e., an open slot and the template or component containing that open slot) in which this building block is to be placed. For example, a building block which contains a specific sorting algorithm, may require as its input an unordered list of elements, where each element consists of an unknown part and an explicit key. In addition, the pre-conditions describe which languages are used in the description.

The properties of a building block are divided into properties concerning the templates and components, and properties concerning open slots. Examples of properties of a conceptual template containing a skeleton for an agent are: it is autonomous, it is capable of communicating with other agents, it is capable of interacting in the world, it is capable of retaining information on other agents and the world. Properties of an open slot may be, for example, that a specific open slot contains an agent communication language syntax

expressed in XML. Template properties at a detailed (operational) level include properties such as: an agent is a process, the size is so many bytes, and the datastructure is of a specific class. Properties of open slots are, for example, that the first argument in a specific open slot contains the input-information for a specific process, and the second argument contains a pointer to a data structure of a specific class for the results.

## 3.4 Retrieving building blocks

The agent factory is able to retrieve templates and components on the basis of needs for adaptation. The re-design process inside the agent factory analyses needs for adaptation and transforms these into requirements (on structure, functionality, and behaviour) on agents to be constructed. The agent design is a configuration of templates and components that satisfies these requirements.

Matching requirements on structure, functionality, and behaviour of (parts of) agents to properties of templates and components is not trivial. Requirements may be incomplete, conflicting, or vague. To solve this problem, a matching process is needed which has some understanding of the properties involved.

Properties are related to each other in property networks. This allows generic properties to be, for example, refined into a number of sets of more refined properties. Two assumptions are made: if a more generic property of an agent holds, then at least one set of refined properties holds. If all refined properties of one set hold, then the more generic property also holds.

A number of refinements may exist for a specific property, each of which can be included in a refinement tree. Refinement trees can be combined into property networks. In these networks, it is possible to explore alternative refinements of a property. For example, the property that a specific algorithm is a sorting algorithm can be refined into more specific properties on efficiency, e.g. sorting algorithms in linear time, in $O(^n log(n))$ time, etc. Alternatively, the 'sorting algorithm' property may be refined into more specific properties on the number of keys used: one key, one primary key and one secondary key, etc. Yet another alternative is that this property is, in itself, a refined property of a property expressing that an algorithm is a classification algorithm.

The matching process has variable forms of interpretation. One form is that no interpretation is used at all (syntactical or exact matching), so that a required property needs to be explicitly present in a building block. An alternative is to use property refinement: a high level property (e.g., an algorithm which orders a list of elements) for which no building block can be found, can be refined into a more specific property (e.g., an algorithm which orders an array of elements in $O(n^2)$ time), for which a building block can be found. Usually, a building block will exist with a more specific property, which can then fulfil the desired property.

A more elaborate means of query interpretation is by traversing semantic property networks. This usually returns a 'good guess', but not necessarily an optimal answer as a building block with similar properties is returned. The notion of 'similar' can be tuned (e.g., what distance to travel through property networks).

## 3.5 The process of adaptation

The agent factory is able to adapt an existing agent on the basis of needs for adaptation. The agent factory re-designs agents. The agent factory first obtains an initial set of required properties (the needs for adaptation) and a description of the agent to be adapted.

The initial set of required properties is analysed and manipulated (e.g., interpreted, conflicts are resolved, etc.) to form a set of refined required properties that are still related to the initial set, yet are more specific. This may already involve checking the library of templates and components for the presence of templates and components with specific properties (it makes no sense to require, for example, a sorting algorithm in O(1/n) time if there are no such building blocks in the library).

On the basis of such a more specific set of required properties, the conceptual description of the agent is adapted. Building blocks are inserted, moved, and/or deleted, until the required properties are satisfied if possible. Additional adaptation of the set of required properties may be necessary (if, for example, the required properties prove to be conflicting). A new set of required properties may be constructed, based on the previous set of required properties and evaluations of the success or failure in constructing a satisfactory conceptual description.

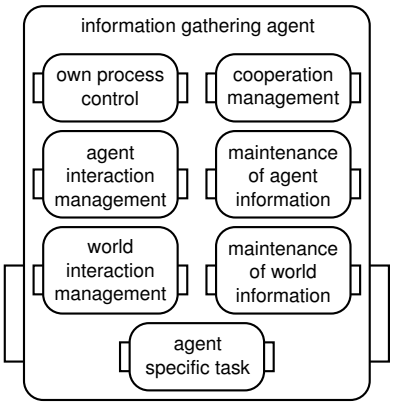


Figure 3: The seven processes inside the information gathering agent. Each process, including the agent itself, has an interface. Between processes, information transfer is defined (not shown).

At some point in this cycle, the conceptual description of an agent is analysed to check whether it satisfies a specific set of required properties (based on the initial set of required properties). If this point has been reached, the agent factory focusses on adapting the detailed, operational description of the agent. If not, the agent factory may adapt the set of required properties.

The operational description of an agent is based on the configuration of templates and components in the conceptual description of the agent. If problems occur in combining operational descriptions from templates and components, either other templates and components are used (with the same conceptual description and properties, but different operational description and properties) or a different conceptual description of the agent is needed. The process described above is then repeated with additional requirements (i.e., required properties).

# 4 Automated servicing of an information gathering agent

In this paper an example is given of an agent that requires servicing. The adaptation of an information gathering agent in this example is based on an existing prototype automated

servicing service. The conceptual descriptions of the templates and components are specified in the DESIRE knowledge-level modelling language (Brazier et al., 1998) and the operational descriptions are in Java.

The information gathering agent used in this example is based on a template containing a generic co-operative agent model (Brazier et al., 1996). Figure 3 illustrates the seven processes distinguished in this generic model. This architecture models an agent that:

- reasons about its own processes (component Own Process Control, or opc),
- communicates with other agents (component Agent Interaction Management, or aim),
- maintains information about other agents (component Maintenance of Agent Information, or mai),
- interacts with the external world (component World Interaction Management, or wim),
- maintains information about the external world (component Maintenance of World Information, or mwi),
- participates in project co-ordination (component Co-operation Management, or cm) and
- the agent's specific tasks (component Agent Specific Tasks, or ast).

This model of a co-operative agent includes components for management of its own processes, interaction with other agents including co-operation, interaction with the external (material) world, and an agent's more specific tasks. In this model, a co-operative agent receives messages from other agents, and observations in the external world (its input). It sends messages to other agents and directs its own observations and actions in the external world (its output).

In this section two examples are given of adaptation of the information gathering agent. In the first example, the information gathering agent is adapted to include functionality for understanding a new language (Section 4.1). In the second example, the information gathering agent is adapted to include new functionality for (more) secure communications and co-operations (Section 4.2).

## 4.1 Shallow adaptation

Figure 4 depicts the information gathering agent and shows that both the agent interaction management process (aim) and the co-operation management process (cm) are open slots. The open slot of the agent interaction management process is filled by two components providing functionality for understanding a communication language with the personal assistant agent and an information provider agent (e.g., which provides access to the World-Wide Web). The open slot of the co-operation management process is filled by two components providing functionality for understanding how to co-operate (protocols) with the personal assistant agent and an information provider agent.

One of the needs for adaptation identified in Section 2 was that the information gathering agent needed to interact with the travel agent. The travel agent was able to indicate that a specific language and protocol was to be employed. One way for the information gathering agent to approach this problem is to use the agent factory to have itself

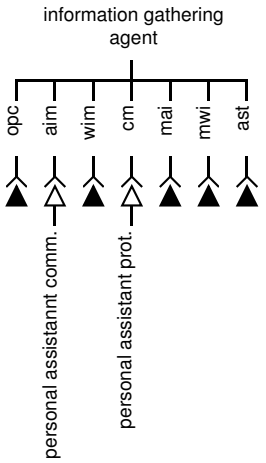


Figure 4: Partial description of the information gathering agent. The open slots for the agent interaction management and co-operation management processes are filled with two components each: agent communication languages and protocols.

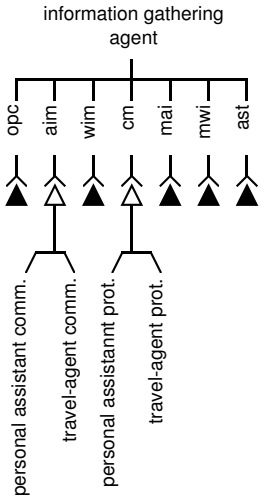


Figure 5: Partial description of the information gathering agent. The open slots for the agent interaction management and co-operation management processes are filled with two components each: agent communication languages and protocols.

changed, such that it can understand the languages and protocols needed for interaction with the travel agent.

In the first case, the agent factory searches its libraries of templates and components and is able to find components that support the functionality required. In addition, these components contain descriptions at both levels of abstraction, and each description needs to be linked to the, already existing, description of the information gathering agent. This results in a description of the information gathering agent, as depicted in figure 5.

The information gathering agent is adapted to include functionality on a language and protocol for interaction with the travel agent.

First the new components are inserted into the conceptual description of the agent. Once this has been achieved successfully, the operational parts of the components are inserted into the operational description of the agent.

## 4.2 Deep adaptation

In another example in Section 2, one of the needs for adaptation arises from communication with a bank agent. This agent requires that the information gathering agent uses specific security functionality. Again, the information gathering agent uses the agent factory to have itself adapted.

The agent factory now has two goals in adapting the information gathering agent. Specific security functionality needs to be added, and functionality for understanding a language and protocol shared with the bank agent. The latter case has been described in the previous subsection.

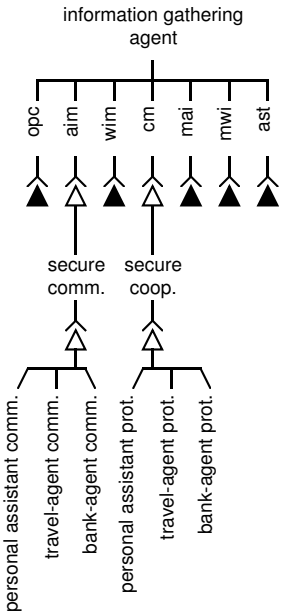


Figure 6: The information gathering agent is adapted to include functionality on secure communication and co-operation, and functionality on understanding a language and protocol for interaction with the bank agent.

Adapting the information gathering agent to include specific security functionality is translated by the agent factory to the need to adapt the agent to include functionality for secure communication and secure co-operation (as in both processes security-related awareness is needed). Two conceptual templates have been retrieved from the library available to the agent factory: a template for secure communication and a template for secure co-operation. Both templates can be used together, as can be derived form their characteristics, and both templates can be embedded in the current configuration of templates and components. Detailed templates are available for these conceptual templates, which can also interface with detailed templates and components in the current detailed configuration of the information gathering agent.

As shown in figure 6, the information gathering agent is modified in a non-trivial manner: the two new templates are inserted into two of the open slots of the top-most template, and the original fillings of these open slots are inserted into the open slots of the new templates.

# 5 Feasibility

The feasibility of an agent factory hinges on a number of aspects. These aspects are briefly described in Section 5.1. A comparison of the agent factory to other approaches in constructing agents in described in Section 5.2.

## 5.1 Crucial aspects

A number of aspects are crucial to the feasibility of an agent factory. These aspects are mainly related to building blocks; the templates and components. Inserting templates or components into an open slot of a template or component involves understanding:

- the properties associated with the interface required by an open slot, which prescribe properties of the interfaces of entities to be inserted,
- how properties relate to each other,
- how a description of the template or component to-be-inserted, can be connected to an open slot (this may involve a mapping of interfaces expressed in two different conceptual description languages),
- how multiple components can be inserted into one open slot (cf. to stacking blocks), especially when different description languages are employed.

Experience with the current prototype has increased confidence in the feasibility of the agent factory. This prototype is capable of automatically configuring relatively simple information retrieval agents from a limited set of building blocks. An agent can be constructed and/or adapted, on the basis of a description of required functionality. This prototype uses a framework for describing conceptual descriptions based on DESIRE (Brazier et al., 1998) (simplified) and operational descriptions based on the programming language Java. The performance of the current prototype is limited in both functionality and resource usage. Current and future research focusses more on improving the functionality of the agent factory than reducing its resource usage.

More research is needed (and is being conducted) to, e.g., develop ontologies for building blocks, extend the library with building blocks for other types of agents, and assess genericity and specificity of (descriptions of) building blocks. The use of additional

frameworks (such as UML) and languages (such as C) is also being pursued. Current research includes development of a language to describe blueprints (including the configuration of building blocks).

An application area in which the agent factory can play an important role is *g*enerative migration(Brazier et al., 2002). In most of today's agent systems migration of an agent requires homogeneity in the programming language and/or agent platform in which an agent has been designed. The agent factory supports generative migration: agents can migrate between non-identifical platforms and need not be written in the same language. Instead of migrating the 'code' (including data and state) of an agent, a blueprint of the agent is transferred. An agent factory generates new code on the basis of this blueprint. A prototype is currently being developed as a service of agent-oriented middleware: AgentScape.

## 5.2 Comparison to other approaches

The agent factory is in some ways comparable to component-based development, agent construction kits, software reusability, case-based reasoning, configuration design, and IBROW.

The agent factory's approach to combining templates and components seems similar to the approach taken in *component-based development* of software (Hopkins, 2000; Sparling, 2000). One distinction is that our approach includes annotations of templates and components at two levels of abstraction (conceptual and operational). In component-based development, interfaces are described for components (which are independent of an operational language); this corresponds to the descriptions of interfaces of templates and components and interfaces needed by open slots in templates and components. From our perspective component-based development provides a useful means to describe operational descriptions of the building blocks used by the agent factory.

Currently a relatively large number of tools and/or frameworks exists for the (usually semi-automatic) *creation of agents* (not automated adaptation). Examples include e.g. AgentBuilder (Reticular, 1999), D'agents/AgentTCL (Gray et al., 1997), ZEUS (Nwana et al., 1999), NOMADS (Suri et al., 2000), Sensible Agents (Barber et al., 2001), and Tryllian (Tryllian, 2001). All of these approaches commit to a specific operational description of agents, and in some cases also commit to a specific conceptual description of their agents. The agent factory does not make such commitments, making the agent factory more general purpose (with all the common advantages and disadvantages).

The agent factory currently pragmatically circumvents a number of issues related to *software reusability* (e.g., (Biggerstaff and Perlis, 1997)). A major problem is annotating reusable pieces of software such that they can be retrieved at a later time (by other people) and reused with a minimal number of changes. In the agent factory the latter is endeavoured as well. The former is currently solved in a pragmatic way: templates and components are annotated, and, when needed, a mapping is provided to other annotations. This, however, is not a scalable solution, and, as such, is one of our current foci of research. An important decision concerning standardisation is that the agent factory does not aim to adhere to one specific standard, but a number of standards.

In *case-based reasoning* approaches (e.g., (Kolodner, 1993; Watson and Marir, 1994)) libraries of cases are consulted to find a case which matches a problem, upon which retrieved cases are adapted. This approach differs from the agent factory in that cases are modified internally, instead of combined with other cases. Techniques for retrieving cases from case libraries are, of course, relevant to retrieving templates and components from libraries.

The approaches taken by *design-as-configuration* (e.g., as described in (Stefik, 1995), CommonKads (Schreiber et al., 1999), and elevator configuration (Schreiber and Birmingham, 1996)) focus on constructing a satisfactory configuration of elements on the basis of a given set of requirements (also named: constraints). In most of these approaches no explicit manipulation of requirements is present, nor is a multi-levelled description of the elements taken into account. Models and theories on configuration-based design are relevant to the agent factory, in particular to the processes involved in combining conceptual and operational descriptions.

The approach taken is similar, to some extent, to approaches such as *IBROW* (Motta et al., 1999). In IBROW semi-automatic configuration is supported of intelligent problem solvers. Their building blocks are 'reusable components', which are not statically configured, but dynamically 'linked' together by modelling each building block as a CORBA object. The CORBA-object provides a wrapper for the actual implementation of a reusable component. A Unified Problem-solving method development language UPML (Fensel et al., 2001) has been proposed for the conceptual modelling of their building blocks. The agent factory differs in a number of aspects, which include: multiple conceptual and detailed languages, no pre-defined wrappers for detailed building blocks, agents consist of one process, and the process of reconfiguration is an automated (re-)design process.

# 6 Discussion

An automated servicing process for agent adaptation is described in this paper. This servicing process is the task of an agent factory. Agents are constructed from templates and components. Adapting an agent entails adapting the configuration of templates and components.

Five assumptions underly our approach: (1) agents have a compositional structure, (2) re-usable parts of agents can be identified, (3) two levels of descriptions are used: conceptual and operational, (4) properties and knowledge of properties and interfaces of re-usable parts of agents are available, and (5) no commitments are made to specific languages and/or ontologies.

The main advantage of an agent factory as an automated servicing process is that an agent can easily obtain new functionality, without obliging the agent itself to have its own adaptation mechanism. During their lifetime agents acquire new skills and knowledge.

The agent factory is still being researched; the current research focusses on:

- building a library of templates and components,
- designing description languages for properties of interfaces of, and knowledge on the use of, templates and components,
- learning from experiences with different conceptual and operational description languages,
- designing and implementing more extensive prototypes of the agent factory,
- investigating security and trust in using an agent factory.

# Acknowledgements

The authors wish to thank the graduate students Hidde Boonstra and David Mobach for their explorative work on the application of an agent factory for an information retrieving

agent. This work was supported by NLnet Foundation, http://www.nlnet.nl/.

## References

Barber, K., McKay, R., MacMahon, M., Martin, C., Lam, D., Goel, A., Han, D., and Kim, J. (2001). Sensible agents: An implemented multi-agent system and testbed. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-2001)*, pages 92–99.

Benjamins, V. (1995). Problem-solving methods for diagnosis and their role in knowledge acquisition. *International Journal of Expert Systems: Research & Applications*, 8(2):93–120.

Biggerstaff, T. and Perlis, A., editors (1997). *Software Reusability: Concepts and models*, volume 1. New York, ACM Press.

Booch, G. (1991). *Object oriented design with applications*. Redwood City, Benjamins Cummins Publishing Company.

Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D. (2000). Simple object access protocol (soap) 1.1. Technical report, W3C. http://www.w3.org/TR/SOAP/.

Bray, T., Paoli, J., Sperberg-McQueen, C., and Maler, E. (2000). Extensible markup language (xml) 1.0 2nd ed. Technical Report 20001006, W3C. http://www.w3.org/TR/2000/REC-xml-20001006.

Brazier, F., Jonker, C., and Treur, J. (1998). Principles of compositional multi-agent system development. In Cuena, J., editor, *Proceedings of the 15th IFIP World Computer Congress, WCC'98, Conference on Information Technology and Knowledge Systems, IT&KNOWS'98*, pages 347–360.

Brazier, F., Jonker, C., and Treur, J. (2000a). Compositional design and reuse of a generic agent model. *Applied Artificial Intelligence Journal*, 14:491–538.

Brazier, F., Jonker, C., Treur, J., and Wijngaards, N. (2000b). Deliberate evolution in multi-agent systems. In Gero, J., editor, *Proceedings of the Sixth International Conference on AI in Design, AID'2000*, pages 633–650. Kluwer Academic Publishers.

Brazier, F., Jonker, J., and Treur, J. (1996). Modelling project coordination in a multi-agent framework. In *Proc. Fifth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE'96*, pages 148–155. Los Alamitos, IEEE Computer Society Press.

Brazier, F., Overeinder, B., van Steen, M., and Wijngaards, N. (2002). Agent factory: Generative migration of mobile agents in heterogeneous environments. In *Proceedings of the AIMS workshop at SAC-2002*. to appear.

Bui, H., Kieronska, D., and Venkatesh, S. (1996). Learning other agents' preferences in multiagent negotiation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-96)*, pages 114–119.

Fensel, D., Horrocks, I., van Harmelen, F., Decker, S., Erdmann, M., and Klein, M. (2000). Oil in a nutshell. In Dieng, R., editor, *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modelling, and Management (EKAW'00)*, volume 1937 of *Lecture Notes in Artificial Intelligence*, pages 1–16. Springer-Verlag.

Fensel, D., Motta, E., Benjamins, V., Crubezy, M., Decker, S., Gaspari, M., Groenboom, R., Grosso, W., van Harmelen, F., Musen, M., Plaza, E., Schreiber, A., Studer, R., and Wielinga, B. (2001). The unified problem-solving method development language UPML. *Knowledge and Information Systems*. to appear.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of reusable object-oriented software*. Addison Wesley Longman, Reading, Massachusetts.

Gray, R., Kotz, D., Cybenko, G., and Rus, D. (1997). *Agent Tcl*, chapter 4, pages 58–95. Manning Publishing. W. Cockayne and M. Zyda, editor.

Hopkins, J. (2000). Component primer. *Communications of the ACM*, 43(10):27–30.

Horrocks, I., van Harmelen, F., Patel-Schneider, P., Berners-Lee, T., Brickley, D., Connoly, D., Dean, M., Decker, S., Fensel, D., Hayes, P., Heflin, J., Hendler, J., Lassila, O., McGuinness, D., and Stein, L. (2001). Daml+oil. Technical report, DAML. http://www.daml.org/2001/03/daml+oil-index.html.

Jackson, M. (1975). *Principles of Program Design*. Academic Press.

Jackson, M. (1995). *Software Requirements and Specifications*. Addison-Wesley, Wokingham, England.

Kolodner, J. (1993). *Case-Based Reasoning*. Morgan Kauffman, San Mateo, California.

Motta, E., Fensel, D., Gaspari, M., and Benjamins, V. (1999). Specifications of knowledge component reuse. In *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering (SEKE-99)*, pages 17–19, Kaiserslautern, Germany.

Nwana, H., Ndumu, D., Lyndon, L., and Collis, J. (1999). Zeus: A toolkit and approach for building distributed multi-agent systems. In *Proceedings of the Third International Conference on Autonomous Agents (Autonomous Agents'99)*, pages 360–361.

Peña-Mora, F. and Vadhavkar, S. (1996). Design rationale and design patterns in reusable software design. In Gero, J. and Sudweeks, F., editors, *Artificial Intelligence in Design (AID'96)*, pages 251–268, Dordrecht, The Netherlands. Kluwer Academic Publishers.

Pressman, R. (1997). *Software Engineering: A practitioner's approach*. Computer Science. McGraw-Hill, fourth edition.

Reticular (1999). *AgentBuilder: An integrated toolkit for constructing intelligent software agents*. Reticular Systems Inc, white paper edition. http://www.agentbuilder.com.

Riel, A. (1996). *Object-Oriented Design Heuristics*. Addison Wesley Publishing Company, Reading Massechusetts.

Rumbaugh, J., Jacobson, I., and Booch, G. (1999). *The unified modeling language reference manual*. Addison Wesley, Reading, Massachusetts.

Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., de Velde, W. V., and Wielinga, B. (1999). *Knowledge Engineering and Management, the CommonKADS Methodology*. MIT press.

Schreiber, G. and Birmingham, W. (1996). Special issue on sisyphus-vt. *International Journal of Human-Computer Studies (IJHCS)*, 44. editors.

Soltysiak, S. and Crabtree, B. (1998). Knowing me, knowing you: Practical issues in the personalisation of agent technology. In *Proceedings of the third international conference on the practical applications of intelligent agents and multi-agent technology (PAAM98)*, pages 467–484, London.

Sparling, M. (2000). Lessons learned through six years of component-based development. *Communications of the ACM*, 43(10):47–53.

Stefik, M. (1995). *Introduction to Knowledge Systems*. Morgan Kaufmann Publishers Inc., San Francisco, California.

Suri, N., Bradshaw, J., Breedy, M., Groth, P., Hill, G., Jeffers, R., Mitrovich, T., Pouliot, B., and Smith, D. (2000). Nomads: Toward a strong and safe mobile agent system. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 163–164. ACM Press.

Tryllian (2001). Agent development kit. Technical report, Tryllian. http://www.tryllian.com/sub_documentation/whitepapers/english/Technical white paper ADK v1.0.pdf.

Watson, I. and Marir, F. (1994). Case-based reasoning: a review. *The Knowledge Engineering Review*, 9(4):327–354.

Wells, N. and Wolfers, J. (2000). Finance with a personalized touch. *Communications of the ACM, Special Issue on Personalization*, 43(8):31–34.

Wieringa, R. (1996). *Requirements Engineering: Frameworks for Understanding*. Wiley and Sons.

Wong, H.-C. and Sycara, K. (2000). A taxonomy of middle-agents for the internet. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS'2000)*, pages 465–466.